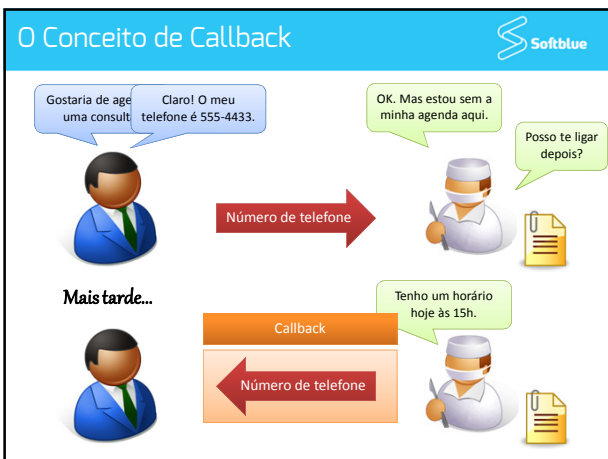




Tópicos Abordados

- O conceito de callback
- Delegates
 - Definindo e utilizando delegates
 - Delegates que referenciam vários métodos
 - Delegates genéricos
 - *Action<>* e *Func<>*
- Events
 - Definindo e registrando interesse
 - Padrão para definição de events
 - Delegate *EventHandler<T>*



O Conceito de Callback

- Em programação, existem várias técnicas para registro de callback
 - Ponteiro para função (C/C++)
 - Referência a um objeto (Java)
 - Delegate (C#)

Delegates

- Um delegate é um objeto capaz de referenciar um método (ou “apontar para um método”)
 - Conceito muito semelhante ao ponteiro para função do C/C++, mas garante segurança na tipagem dos dados
- Um delegate é definido como um tipo de dado em C#
 - Internamente, o compilador cria uma classe para o delegate, que herda de *System.MulticastDelegate*


Criando Delegates

- A criação do delegate é feita através da palavra-chave *delegate*

```
delegate void AvisoConsulta(DateTime horario, string obs);
```

O delegate *AvisoConsulta* pode apontar para qualquer método que respeite a sua assinatura

Referenciando um Método



```
delegate void AvisoConsulta(DateTime horario, string obs);

static void MostrarConsulta(DateTime horario, string obs)
{
    Console.WriteLine(horario);
    Console.WriteLine(obs);
}


AvisoConsulta callback = new AvisoConsulta(MostrarConsulta);
```

O método a ser referenciado é passado como parâmetro no construtor do delegate

Um delegate só pode apontar para métodos que tenham assinatura compatível com a sua

Podem ser usados métodos estáticos ou de instância

Chamando o Callback




- O objeto do delegate deve ser usado para fazer a chamada
 - É como se a chamada fosse feita diretamente ao objeto
 - O delegate se encarrega de encaminhar a chamada para o método referenciado

```
AvisoConsulta callback = new AvisoConsulta(MostrarConsulta);
callback(DateTime.Now, "Chegar com 10min de antecedência");
```

O delegate vai chamar o método *MostrarConsulta()*

Múltiplos Callbacks



- O delegate também é capaz de referenciar mais de um método

```
static void MostrarConsulta(DateTime horario, string obs) { }
static void AgendarConsulta(DateTime horario, string obs) { }

AvisoConsulta callbacks = new AvisoConsulta(MostrarConsulta);
callbacks += new AvisoConsulta(AgendarConsulta);

callbacks(DateTime.Now, "Chegar com 10min de antecedência");
```

O operador += é usado para registrar mais de um método

O operador -= tem o papel inverso

MostrarConsulta()
AgendarConsulta()

Simplificando a Criação de um Delegate



- O objeto delegate pode ser substituído apenas pelo nome do método que ele referencia

```
AvisoConsulta callback = new AvisoConsulta(MostrarConsulta);  
RegistrarCallback(callback);
```



```
RegistrarCallback (MostrarConsulta);
```

```
AvisoConsulta callbacks = new AvisoConsulta(MostrarConsulta);  
callbacks += new AvisoConsulta(AgendarConsulta);
```



```
AvisoConsulta callbacks = MostrarConsulta;  
callbacks += AgendarConsulta;
```

Delegates Genéricos



- Um delegate costuma ser usado apenas em um contexto específico
 - Baixo reaproveitamento
- Isto ocasiona a criação de uma série de delegates no código da aplicação
- Delegates genéricos podem ser utilizados para evitar este problema
 - Usados quando o nome do delegate não importa
- Podem ser de dois tipos
 - *Action<>*
 - *Func<>*

Delegate *Action<>*



- Pode referenciar métodos de até 16 parâmetros
- Os métodos devem ter retorno *void*

```
Action<DateTime, string> callback =  
new Action<DateTime, string>(MostrarConsulta);
```

```
Action<DateTime, string> callback = MostrarConsulta;
```

Delegate Func<>



- Pode referenciar métodos de até 16 parâmetros
- Os métodos podem ter retorno
 - O último tipo parametrizado fornecido é o tipo do retorno

```
Func<int, string, bool> callback =  
    new Func<int, string, bool>(MeuMetodo);  
  
Func<int, string, bool> callback = MeuMetodo;  
  
public static bool MeuMetodo(int i, string s) { }
```

Events



- Events representam eventos
 - Um objeto pode avisar a respeito de eventos que ocorrem com ele
 - Outros objetos que registram interesse no evento em questão, são avisados
- Events precisam de delegates para funcionarem
 - É possível ter o mesmo comportamento apenas usando delegates
 - No entanto, o código fica mais complexo

Criando Events



- Criação do delegate
- Criação da classe e do event

```
delegate void SapoHandler(double distancia);
```

```
class Sapo  
{  
    public event SapoHandler Pulou;  
  
    public void Pular()  
    {  
        Random r = new Random();  
        int distancia = r.Next(30);  
  
        if (Pulou != null)  
        {  
            Pulou(distancia);  
        }  
    }  
}
```

Definição
do event

Disparo do
event

Registrando o Interesse em Events

- O objeto interessado no evento faz o registro do callback

```

class Pesquisador
{
    public Pesquisador(Sapo sapo)
    {
        sapo.Pulou += SapoPulou;
    }

    public void SapoPulou(double distancia)
    {
        //...
    }
}
        
```

Registra interesse no event
Pulou do objeto *Sapo*

Quando o evento ocorrer,
SapoPulou será chamado

O operador "+=" pode ser usado para remover o interesse em um event

Padrão para Definição de Events

- A Microsoft recomenda que eventos chamem métodos que:
 - Recebam dois parâmetros
 - object sender*
 - System.EventArgs args*
 - Retornem *void*
- O primeiro parâmetro indica o objeto que disparou o evento
- O segundo parâmetro define as informações do evento

Exemplo de Uso do Padrão

- Classe para as informações do event

```

class PulouEventArgs : EventArgs
{
    public readonly double Distancia;

    public PulouEventArgs(double distancia)
    {
        this.distancia = distancia;
    }
}
        
```


Herda de *System.EventArgs*

- Definição do delegate

```

delegate void SapoHandler(object sender, PulouEventArgs args);
        
```

Exemplo de Uso do Padrão



```

class Sapo
{
    public event SapoHandler Pulou;

    public void Pular()
    {
        Random r = new Random();
        int distancia = r.Next(30);
        if (Pulou != null)
        {
            Pulou(this, new PulouEventArgs(distancia));
        }
    }
}

```

Dispara o evento fornecendo os dois parâmetros

```


class Pesquisador
{
    public Pesquisador(Sapo sapo)
    {
        sapo.Pulou += SapoPulou;
    }

    public void SapoPulou(object sender, PulouEventArgs args)
    {
        Sapo s = sender as Sapo;
        double d = args.Distancia;
        //...
    }
}

```

O método chamado deve ter a assinatura correta

O Delegate *EventHandler<T>*



- Muitos eventos recebem como parâmetro um *object* e um *EventArgs*
- O delegate *EventHandler<T>* facilita o uso destes tipos de eventos

```

class Sapo
{
    public event EventHandler<PulouEventArgs> Pulou;
    ...
}

```

T define a classe que herda de EventArgs

```

class Pesquisador
{
    public void SapoPulou(object sender, PulouEventArgs args) { }
    ...
}

```

O event pode chamar métodos com os parâmetros (object, PulouEventArgs)

Não é mais necessário definir um delegate customizado